Минобрнауки России

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Работа допущена к защите

И. о. заведующего кафедрой КИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Щукин

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**ПРИМЕНЕНИЕ ИСКУССТВЕННОЙ НЕЙРОННОЙ СЕТИ ДЛЯ ОЦЕНКИ РИСКА ЗАРАЖЕНИЯ КОМПЬЮТЕРА ВРЕДОНОСНЫМ ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ**

по направлению 09.03.03 Прикладная информатика

по образовательной программе

09.03.03\_03 Прикладная информатика в области информационных ресурсов

Выполнил

студент гр.43506/3 А.В. Заднепровский

Руководитель

доцент, к. ф.-м. н. В.Г. Пак

Консультант

по нормоконтролю О.В. Колосова

Санкт-Петербург

2019

**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО**

**ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ**

УТВЕРЖДАЮ

И.о. заведующего кафедрой КИТ

А.В. Щукин

« » 2019 г.

ЗАДАНИЕ

по выполнению выпускной квалификационной работы

студенту

Заднепровскому Андрею Владимировичу 43506/3

1. Тема работы: «Применение искусственной нейронной сети для оценки риска заражения компьютера вредоносным программным обеспечением».

2. Срок сдачи студентом законченной работы: май 2019.

3. Исходные данные по работе: Microsoft Malware Prediction Dataset, модели ИНС, применимые для решения задачи предсказания, алгоритмы обучения ИНС.

4. Содержание работы (перечень подлежащих разработке вопросов):

1. Постановка задачи, обоснование актуальности;
2. Обзор моделей ИНС, применимых для решения задачи, алгоритмов обучения, существующих решений;
3. Выбор и разработка архитектуры ИНС, применительно к задаче, с обоснованием, описание архитектуры, алгоритмов обучения;
4. Анализ и предварительная обработка исходных данных;
5. Реализация ИНС;
6. Настройка гиперпараметров модели ИНС;
7. Экспериментальное исследование эффективности разработанной ИНС;
8. Сравнение с существующими решениями;
9. Выводы об эффективности разработанной ИНС, перспективы улучшения.

5. Перечень графического материала (с указанием обязательных чертежей): графики.

6. Консультанты по работе:

7. Дата выдачи задания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) инициалы, фамилия

Задание принял к исполнению

(дата)

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) инициалы, фамилия

# РЕФЕРАТ

46 с., 14 рисунков, 2 таблицы, 3 приложения.

МАШИННОЕ ОБУЧЕНИЕ, ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, БИНАРНАЯ КЛАССИФИКАЦИЯ, БОЛЬШИЕ ДАННЫЕ, ВРЕДОНОСНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Данная работа относится к сфере машинного обучения. В первой главе ставится задача оценки рисков заражения компьютера вредоносным ПО и делается обзор подходов, применимых для решения этой задачи. Во второй и третьей главах разрабатывается и реализуется модель ИНС, производится первичная оценка ее эффективности для решения поставленной задачи. В четвертой главе производится тестирование модели ИНС при различных наборах гиперпараметров, в результате которого в финальную модель ИНС вносятся необходимые улучшения. По итогам работы делается вывод об эффективности полученной ИНС и перспективах ее улучшения.

# THE ABSTRACT

46 p., 14 pictures, 2 tables, 3 applications.

MACHINE LEARNING, ARTIFICAL NEURAL NETWORK, BINARY CLASSIFICATION, BIG DATA, MALWARE

This work belongs to the sphere of machine learning. The first chapter sets the task of assessing the risk of a computer being infected with malware and reviews approaches applicable to this task. In the second and third chapters, the ANN model is developed and implemented, the initial assessment of its effectiveness for solving the problem is carried out. The fourth chapter tests the ANN model with various sets of hyperparameters, as a result of which the necessary improvements are made to the final ANN model. Based on the results of the work, a conclusion is made about the effectiveness of the ANN and the prospects for its improvement.

# СОДЕРЖАНИЕ

[Введение 6](#_Toc9888831)

[Глава 1. Описание задачи и обзор подходов для ее решения 9](#_Toc9888832)

[1.1. Задача оценки рисков заражения компьютера вредоносным ПО 9](#_Toc9888833)

[1.2. Метрика для оценки эффективности решения 9](#_Toc9888834)

[1.3. Подходы, применяющиеся для решения задачи 11](#_Toc9888835)

[1.3.1. Деревья решений 12](#_Toc9888836)

[1.3.2. Случайные леса 15](#_Toc9888837)

[1.3.3. Градиентный бустинг 17](#_Toc9888838)

[1.3.4. Логистическая регрессия 17](#_Toc9888839)

[1.3.5. Искусственные нейронные сети 19](#_Toc9888840)

[1.4. Выводы по первой главе 21](#_Toc9888841)

[Глава 2. Разработка модели инс 22](#_Toc9888842)

[2.1. Анализ исходных данных 22](#_Toc9888843)

[2.1.1. Обнаружение избыточных признаков 23](#_Toc9888844)

[2.1.2. Подготовка данных 24](#_Toc9888845)

[2.2. Типы слоев 26](#_Toc9888846)

[2.2.1. Вычислительные слои 26](#_Toc9888847)

[2.2.2. Слои преобразования данных 27](#_Toc9888848)

[2.3. Функция потерь 29](#_Toc9888849)

[2.4. Оптимизатор 29](#_Toc9888850)

[2.5. Топология сети 30](#_Toc9888851)

[Глава 3. Програмная реализация инс 32](#_Toc9888852)

[3.1. Используемые программные средства 32](#_Toc9888853)

[3.1. Предварительная обработка данных 33](#_Toc9888854)

[3.3. Реализация ИНС 34](#_Toc9888855)

[Глава 4. Исследование эффективности ИНС 37](#_Toc9888856)

[4.1. Подбор размера полносвязных слоев и глубины ИНС 38](#_Toc9888857)

[4.2. Применение слоев активации и нормализации 39](#_Toc9888858)

[4.3. Применение регуляризации и прореживания 37](#_Toc9888859)

[Заключение 40](#_Toc9888860)

[Список использованных источников 41](#_Toc9888861)

[Приложение 1 43](#_Toc9888862)

[Приложение 2 47](#_Toc9888863)

[Приложение 3 51](#_Toc9888864)

# ВВЕДЕНИЕ

Искусственные нейронные сети (ИНС) являются одним из методов машинного обучения, позволяющим «справляться с задачами, которые слишком сложны для решения с помощью фиксированных программ, спроектированных и написанных людьми». [1, с. 97] Характерной чертой алгоритмов машинного обучения является процедура обучения, а не прямое решение задачи. В ходе обучения происходит тренировка алгоритма на большом количестве примеров решения конкретной задачи.

В настоящее время ИНС находится в наиболее активной фазе развития за всю свою историю, хотя ключевые алгоритмы были уже известны в 90-х годах ХХ века. Это связано с влиянием следующих факторов:

* Значительным приростом вычислительной мощности оборудования, так как обучение ИНС очень требовательно к вычислительным ресурсам компьютера;
* Накоплением больших объемов данных, благодаря бурному росту сети Интернет и емкости устройств хранения информации.

Также было разработано и много новых алгоритмов, однако этого бы не случилось без практических экспериментов, которые стали возможными благодаря описанным выше причинам. [2, c. 45]

За последнее время число задач, для которых применяются ИНС значительно возросло, это стало возможным за счет демократизации инструментов разработки, то есть появления высокоуровневых фреймворков для реализации моделей ИНС.

Одной из новых задач, для решения которой можно применить нейронные сети, является оценка рисков заражения компьютера вредоносным программным обеспечением (ПО) по его характеристикам. Данная задача актуальна, так как разработка вредоносного ПО продолжает оставаться организованной и хорошо финансируемой сферой, которая наносит значительный вред мировой экономике, исчисляющийся в миллиардах долларов.

Объектом исследования в данной работе являются искусственные нейронные сети.

Предмет исследования – методы и алгоритмы применения искусственной нейронной сети в задаче оценки вероятности заражения компьютера вредоносным ПО.

Целью дипломной работы является разработка и обучение ИНС, способной эффективно оценивать вероятность заражения компьютера вредоносным ПО. После тестирования реализованной ИНС требуется сделать выводы о ее эффективности по сравнению с существующими решениями, и предложить возможные способы ее улучшения. Для достижения данной цели были выделены следующие задачи:

1. Исследовать методы машинного обучения, которые в настоящее время применяются для задачи оценки вероятности заражения компьютера;
2. Разработать архитектуру ИНС;
3. Реализовать ИНС;
4. Протестировать полученную сеть и добиться улучшения ее эффективности, подобрав оптимальную комбинацию гиперпараметров;
5. Сделать вывод об эффективности реализованной сети по сравнению с существующими решениями, и предложить возможные способы ее улучшения.

Гипотеза исследования: ИНС способна эффективно справиться с задачей оценки вероятности заражения компьютера вредоносным ПО.

В соответствии с намеченной целью и задачами исследования были определены следующие методы:

* теоретический анализ научной и методической литературы по теме исследования;
* изучение документации необходимых языков программирования и библиотек;
* экспериментальный метод, включающий проведение констатирующего эксперимента, анализ и обобщение полученных данных.

Практическая значимость исследования, проведенного в дипломной работе, состоит в реализации эффективного решения задачи оценки вероятности заражения компьютера вредоносным ПО с помощью ИНС.

# ГЛАВА 1. ОПИСАНИЕ ЗАДАЧИ И ОБЗОР ПОДХОДОВ ДЛЯ ЕЕ РЕШЕНИЯ

## 1.1. Задача оценки рисков заражения компьютера вредоносным ПО

В 2018 году компания Microsoft предоставила научному сообществу беспрецедентный набор данных о вредоносных программах, чтобы стимулировать прогресс в разработке эффективных методов прогнозирования заражения компьютера вредоносными программами. Эти данные были предоставлены в рамках соревнования Microsoft Malware Prediction Challenge для специалистов в области анализа данных. [3]

Цель соревнования заключается в разработке эффективного метода предсказания вероятности заражения компьютера под управлением ОС Windows, основываясь на различных свойствах компьютера, например, версии ОС, версии антивируса, наличии межсетевого экрана, производителе оборудования ПК и т.д. Всего набор содержит 82 характеристики для каждого ПК.

Описанная задача относится к категории симметричной бинарной классификации, решением которой должна быть непрерывная величина от 0 до 1, являющаяся степенью уверенности в том, что данный компьютер будет заражен в будущем.

## 1.2. Метрика для оценки эффективности решения

Для оценки эффективности полученных решений в рамках данной работы будет использоваться метод подсчета площади под кривой рабочей характеристики приемника или ROC-кривой (Area Under Receiver Operating Characteristic Curve). Данный метод находит свое применение с 40-х годов ХХ века, и в настоящее время продолжает широко применяться для сравнения алгоритмов бинарной классификации.

Рассмотрим на примере работу данного метода. Пусть некоторый алгоритм классификации выдал оценки, как показано в таблице 1.1. Упорядочим строки таблицы 1.1 по убыванию ответов алгоритма – получим таблицу 1.2.

Таблица 1.1

Пример оценок некоторого алгоритма классификации

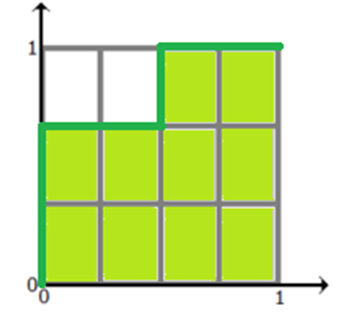
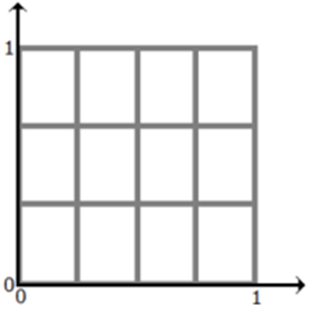
|  |  |  |
| --- | --- | --- |
| Номер | Оценка | Класс |
| 1 | 0.1 | 0 |
| 2 | 0.9 | 1 |
| 3 | 0.3 | 1 |
| 4 | 0.2 | 0 |
| 5 | 0.5 | 0 |
| 6 | 0.4 | 0 |
| 7 | 0.8 | 1 |

Таблица 1.2

Упорядоченные оценки некоторого алгоритма классификации

|  |  |  |
| --- | --- | --- |
| Номер | Оценка | Класс |
| 2 | 0.9 | 1 |
| 7 | 0.8 | 1 |
| 5 | 0.5 | 0 |
| 6 | 0.4 | 0 |
| 3 | 0.3 | 1 |
| 4 | 0.2 | 0 |
| 1 | 0.1 | 0 |

Чтобы нарисовать ROC-кривую, нужно взять единичный квадрат на координатной плоскости, разбить его на m равных частей горизонтальными линиями и на n – вертикальными, где m – число правильно классифицированных единиц, а n – нулей. В результате квадрат разбивается сеткой на m×n блоков, рисунок 1.1а.



b)

а)

Рисунок 2.1. Построение ROC-кривой

Теперь будем просматривать строки таблице 1.2 сверху вниз и прорисовывать на сетке линии, переходя их одного узла в другой. Начинаем из точки (0, 0). Если значение метки класса в просматриваемой строке 1, то делаем шаг вверх; если 0, то делаем шаг вправо. После прохода по всей таблице получится график, представленный на рисунке 1.1b. Ясно, что в итоге мы попадём в точку (1, 1), т.к. сделаем в сумме m шагов вверх и n шагов вправо. [4]

Площадь под ROC-кривой для этого примера будет равна 10/12 ~ 0.83.

## 1.3. Подходы, применяющиеся для решения задачи

Задача бинарной классификация является классической задачей машинного обучения, для ее решения существуют следующие наиболее распространенные методы:

* деревья решений,
* случайные леса,
* градиентный бустинг,
* логистическая регрессия,
* ИНС.

Далее следует подробный обзор всех представленных выше методов.

### 1.3.1. Деревья решений

Деревья решений — это весьма распространенный метод, который используется для решения разного класса задач:

* Классификации: распределение объектов по заранее определенным классам;
* Регрессии: предсказания целевой переменной, которая имеет непрерывную область значения, например численное прогнозирование;
* Описания данных: для хранения информации в компактной форме и быстрого поиска. [5]

Далее будем говорить о деревьях решений применительно к задаче классификации.

Итак, «дерево решений – это модель, представляющая собой совокупность правил для принятия решений». [6] Графически она может быть представлена в виде дерева (рис. 1.2), в котором моменты принятия решений соответствуют узлам дерева (англ.: decision nodes). В них происходит ветвление процесса (англ.: branching), т.е. деление его на ветви (англ.: branches) в зависимости от сделанного выбора. Конечные узлы называют листьями (англ.: leaf nodes), при этом каждый лист – это конечный результат последовательного принятия решений.

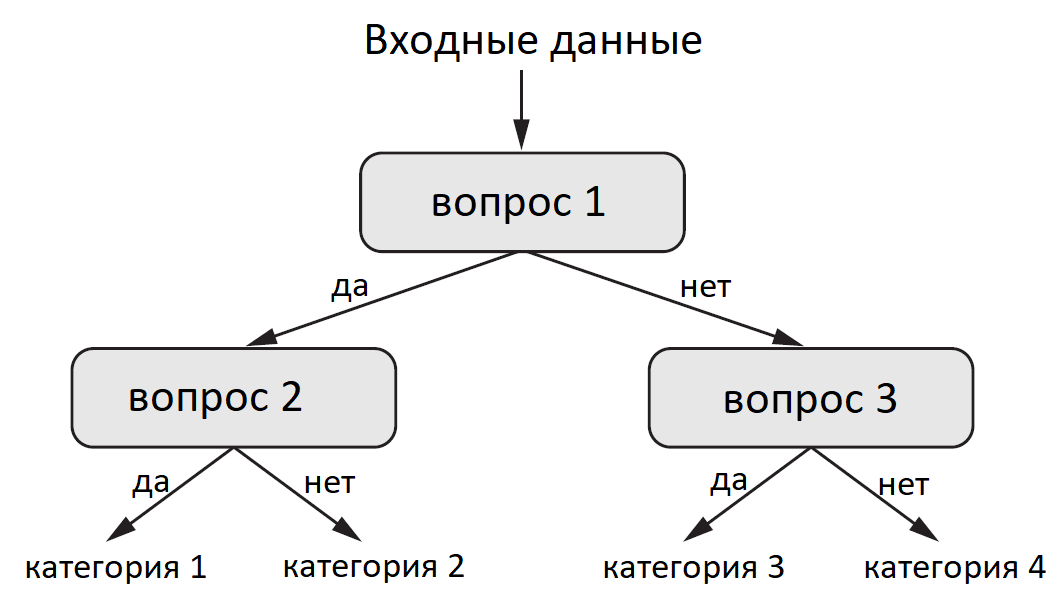


Рисунок 3.2. Визуализация дерева решений

Данные, подлежащие классификации, находятся в корне дерева (англ.: root). При построении дерева в каждом из его узлов выбирается признак, значение которого используется для разбиения всех данных на 2 класса. Условиями для такого разбиения являются обучаемые параметры метода. Пример такого вопроса: признак номер 3 в данных больше 0,5.

Таким образом, для того чтобы построить дерево, мы добавляем все новые узлы, с новыми вопросами о данных, до тех пор, пока не выполнится один из возможных критериев остановки. Это возможно в следующих ситуациях:

* Все, или практически все данные узла принадлежат одному и тому же классу;
* Закончились признаки, по которым можно построить новое разбиение;
* Достигнут лимит роста дерева, если этот параметр был задан. [6]

Проиллюстрируем работу уже построенного решающего дерева высоты 2 на простом примере классификации трех типов объектов (круг, квадрат, треугольник) на координатной плоскости (рис 1.3). Как можно заметить, каждый из вопросов делит оставшееся множество объектов на 2 группы, пока большая часть из них не окажется однотипными.

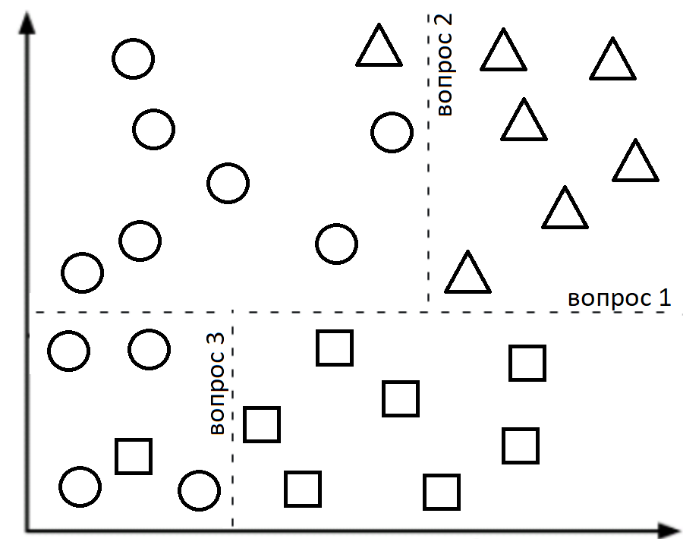


Рисунок 1.3. Иллюстрация работы дерева решений на координатной плоскости

Преимущества метода решающий деревьев:

* Легко визуализируется и интерпретируется;
* Не требует подготовки данных, поддерживает порядковые и номинальные признаки;
* Быстрое обучение и прогнозирование;
* Использует модель «белого ящика». Любое решение можно объяснить с помощь булевой логики;
* Позволяет оценить модель при помощи статистических тестов, что даёт возможность оценить надёжность модели;
* Метод хорошо работает даже в том случае, если были нарушены первоначальные предположения, включённые в модель.

Недостатки метода решающий деревьев:

* Большая чувствительность к шумам во входных данных, «лишние» данные сильно ухудшают точность метода;
* Плохо справляется с разреженными значениями признаков;
* Нестабильность. Небольшие изменения данных приводят к существенным изменениям построенного дерева;
* Задача построения оптимального дерева решений является NP-полной, поэтому для построения деревьев используются жадные алгоритмы, где оптимальное решение выбирается локально в каждом узле. Однако такие алгоритмы не могут обеспечить оптимальность всего дерева в целом;
* Склонность к переобучению, с которым необходимо бороться регулированием максимальной глубины дерева;
* Не умеет экстраполировать, только интерполировать, из-за чего уступает по качеству классификации некоторым другим методам.

Резюмируя, отметим, что в чистом виде деревья решений хоть и показывают неплохую эффективность, особенно в небольших задачах с низкой размерностью пространства признаков, однако они уступают по эффективности более сложным алгоритмам, которые основываются на деревьях решений. О наиболее распространенных из таких алгоритмов речь пойдет ниже.

### 1.3.2. Случайные леса

Алгоритм случайный лес (англ.: Random Forest) предлагает надежный и практичный подход к обучению на основе деревьев решений, его основная идея заключается в использовании большого ансамбля решающих деревьев (это число является параметром метода) с последующим объединением выдаваемых ими результатов. Случайные леса применимы к задачам регрессии, классификации и кластеризации, селекции признаков и т.д. Являются достаточно универсальным алгоритмом, можно сказать, что они почти всегда являются оптимальным алгоритмом для любых задач поверхностного машинного обучения.

«В отличие от классических алгоритмов построения деревьев решений в методе случайных лесов при построении каждого дерева на стадиях расщепления вершин используется только фиксированное число случайно отбираемых признаков обучающей выборки (второй параметр метода) и строится полное дерево (без усечения), т. е. каждый лист дерева содержит наблюдения только одного класса.» [7]

Результат классификации определяется путем голосования полученных классификаторов, определяемых отдельными деревьями. При этом точность ансамблей классификаторов существенно зависит от разнообразия (англ.: diversity) классификаторов, входящих в ансамбль, т. е. от того, насколько их решения взаимозависимы. А именно, чем более разнообразны классификаторы ансамбля, тем выше эффективность классификации. [7]

Достоинства алгоритма «случайный лес»:

* Не подвержен переобучению, что выделяет этот метод среди большинства методов классификации;
* Обеспечивает большую точность, чем просто решающие деревья;
* Есть возможность распараллелить вычисления, что существенно при большом объеме обучающих данных;
* Одинаково хорошо обрабатываются порядковые, номинальные и дискретные признаки;
* Существует естественная оценка вероятности ошибочной классификации – метод Out-Of-Bag, рассчитывающаяся на образцах обучающих данных, не использовавшихся при построении решающего дерева.

Недостатки алгоритма случайный лес:

* Большой объем получающихся моделей;
* В отличие от одного дерева, результаты случайного леса сложнее интерпретировать;
* Для данных, включающих категориальные переменные с различным количеством уровней, случайные леса предвзяты в пользу признаков с большим количеством уровней;
* Метод случайных лесов, так же, как и решающие деревья, не умеет экстраполировать.

На основании всего вышеизложенного, можно сказать, что метод «случайный лес» можно использовать для решения поставленной задачи, однако существуют более интеллектуальные методы, основанные на том же принципе (построения ансамбля решающих деревьев), об одном из которых речь пойдет далее.

### 1.3.3. Градиентный бустинг

Градиентный бустинг (англ.: Gradient Boosting Machine), во многом схож с случайным лесом, этот метод машинного обучения основывается на объединении слабых моделей прогнозирования, чаще всего деревьев решений в ансамбли. «Он использует градиентный бустинг, способ улучшения любой модели машинного обучения путем итеративного обучения новых моделей, специализированных для устранения слабых мест в предыдущих моделях.» [5] Прием градиентного бустинга использующий деревья решений позволяет получить модели, которые как правило превосходят случайные леса при сохранении аналогичных свойств. На данный момент это один из лучших алгоритмов, для решения большинства задач машинного обучения, за исключением задач распознавания образов. [2, с. 41]

Для задачи оценки рисков заражения компьютера метод градиентного бустинга является одним из наиболее оптимальных, и он нашел широкое применение в рамках соревнования Microsoft Malware Prediction Challenge. [3]

### 1.3.4. Логистическая регрессия

Логистическая регрессия (англ.: logistic regression) — «метод построения линейного классификатора, позволяющий оценивать апостериорные вероятности принадлежности объектов классам.» [8] Суть линейной регрессии состоит в анализе связи между несколькими независимыми переменными (называемыми также регрессорами) и зависимой переменной .

Для этого делается предположение, о том, что вероятность наступления события равно:

(1.1)

где ;

– коэффициенты регрессии;

– функция активации (рис. 1.4), так же называемая логистической функцией (или сигмоид).

(1.2)

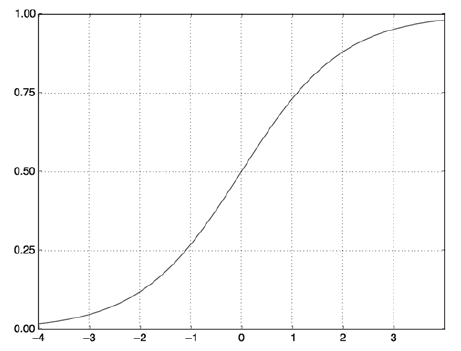


Рисунок 1.4. График логистической функции (sigmoid) [2]

Процесс обучения модели логистической регрессии заключается в подборе коэффициентов регрессии, для этого существует несколько способов. Чаще всего применяется метод максимального правдоподобия, цель которого максимизировать значение функции правдоподобия для обучающей выборки . [9]

(1.3)

Преимущества логистической регрессии:

* Прозрачность моделирования, т. е. доступность для анализа всех промежуточных вычислений;
* Скорость и простота получения модели.

Недостатки логистической регрессии:

* Не подходит для моделирования нелинейных зависимостей;
* Чувствительность к шумам и «лишним» тренировочным данным;
* Чувствительность к избыточным признакам;
* Трудоемкое нахождение коэффициентов регрессии.

Применительно к задаче этой работы данный метод не является подходящим. Это объясняется тем, что алгоритм логистической регрессии относится к категории линейных алгоритмов, которые плохо справляются с задачами, в которых зависимость результата от признаков нелинейная и сложная, т.е. пространство признаков нельзя эффективно разделить гиперплоскостью.

### 1.3.5. Искусственные нейронные сети

Структурно ИНС представляет из себя ориентированный ациклический граф, вершины которого называются нейронами – элементарными единицами обработки информации, а соединительные ребра графа – синапсами, каждый из которых имеет свой вес. Все нейроны делятся на входные, выходные и внутренние (скрытые) (рис 1.5). Исходные данные поступают в ИНС через входные нейроны и далее по синаптическим связями сигнал передается на входы скрытых нейронов.

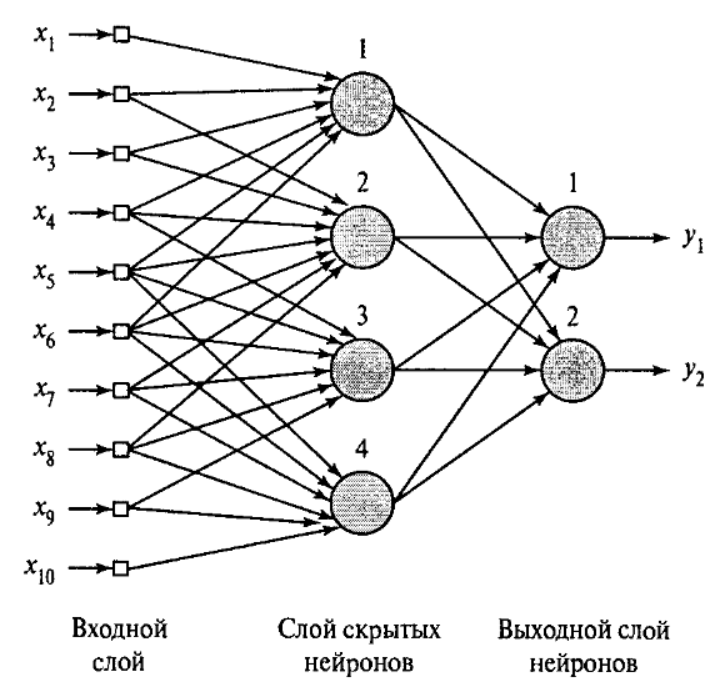


Рисунок 1.5. Пример ИНС [11, с. 64]

Каждый отдельный нейрон вычисляет взвешенную сумму всех входящих сигналов и передает полученное значение по выходным синапсам далее. Сеть заканчивается одним или несколькими выходными нейронами, которые выдают конечный результат работы сети.

Процедура обучения, применительно к ИНС, «выстраивает в определенном порядке синаптические веса нейронной сети для обеспечения необходимой структуры взаимосвязей нейронов». [10] Сеть обучается в результате прохода большого количества циклов обучения, каждый из которых состоит из следующих шагов:

1. Извлекается пакет обучающих экземпляров и соответствующих им целей ;
2. Вычисляются предсказания сети , для обучающих экземпляров ;
3. Оцениваются потери сети на основе расхождений между и ;
4. Корректируются весовые коэффициенты всех синаптических связей так, чтобы незначительно уменьшить потери на этом пакете.

Преимущества ИНС:

* Адаптивность. «Нейронные сети обладают способностью адаптировать свои синаптические веса к изменениям окружающей среды. В частности, для работы в нестационарной среде (где статистика изменяется с течением времени) могут быть созданы нейронные сети, изменяющие синаптические веса в реальном времени»; [10]
* Масштабируемость. Параллельная структура нейронных сетей потенциально ускоряет решение некоторых задач и обеспечивает масштабируемость нейронных сетей;
* Устойчивость к шумам и «лишним» обучающим данным;
* Устойчивость к избыточным признакам;
* Позволяют эффективно строить нелинейные зависимости;
* Может одновременно решать несколько задач на едином наборе входных данных − имея несколько выходов, прогнозировать значения нескольких показателей.

Недостатки ИНС:

* Высокая требовательность к вычислительным ресурсам на стадии обучения сети, однако обученная сеть выдает результат достаточно быстро;
* Необходима тщательная предварительная обработка входных данных;
* Для высокой точности нужен большой объем обучающий данных;
* Работают по принципу «черного ящика», т.е. результаты сети очень трудно интерпретировать.

Основываясь на сказанном выше, стоит отметить, что ИНС хорошо подходят для решения поставленной задачи, из-за способности строить нелинейные зависимости, устойчивости к зашумленным и избыточным данным. Также важна их способности адаптироваться к изменяющейся окружающей обстановке, так как предметная область подразумевает, что с течением времени характер входных данных будет претерпевать значительные изменения.

## 1.4. Выводы по первой главе

В данной главе были исследованы методы машинного обучения, позволяющие решать задачу оценки рисков заражения компьютера вредоносным ПО. На основании чего можно сделать вывод, что для решения поставленной задачи, наряду с градиентным бустингом подходят ИНС. Однако ИНС выглядят предпочтительнее так как:

* на данный момент не нашли широкого применения в рамках решения этой задачи,
* являются одним из самых быстроразвивающихся и популярных методов машинного обучения,
* полученная с его помощью модель имеет способность к адаптации с течением времени.

Это обосновывает применимость и потенциальную эффективность ИНС для решения поставленной задачи.

# ГЛАВА 2. РАЗРАБОТКА МОДЕЛИ ИНС

Разработка любой модели ИНС состоит из следующих ключевых пунктов:

* Выбора типа слоев, которые будут объединяться в сеть;
* Определения функции потерь, значение которой в процессе обучения будет уменьшаться;
* Определения оптимизатора, который определяет способ обучения.

На рисунке 2.1. представлена схематическая взаимосвязь между этими компонентами ИНС. Их выбор напрямую зависит от исходных данных и глобальных целей, которые должна решать сеть. Поэтому на одном из первых этапов конструирования модели ИНС необходимо проанализировать исходные данные предметной области, сконструировать признаки и определить необходимые преобразования.

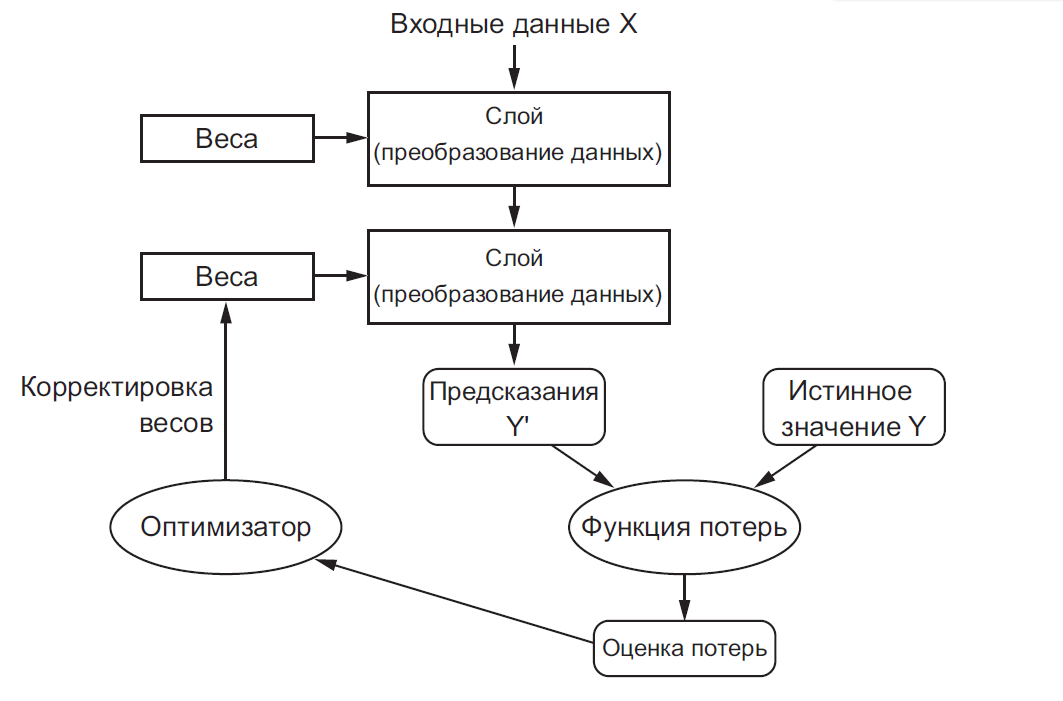


Рисунок 2.1. Связь между сетью, слоями, функцией потерь и оптимизатором [2]

## 2.1. Анализ исходных данных

Исходные данные, представленные компанией Microsoft, содержат информацию о более чем 16 миллионах компьютеров. Каждый компьютер характеризуется 82 признаками (список представлен в приложении 1). Весь объем исходных данных делится на две примерно равные части, это:

* Обучающая выборка, которая содержит в дополнение к 82 характеристикам, дополнительный бинарный признак, говорящий о том, было ли обнаружено заражение вредоносным ПО этого компьютера, или нет;
* Тестовая выборка, которая не содержит правильных ответов, для нее необходимо предсказать ответ. Эффективность классификатора оценивается по тестовой выборке, для этого выданные классификатором оценки отправляются для расчетов на сервер соревнования.

В общем случае, на первом этапе анализа данных следует произвести конструирование признаков. Это такое преобразование исходных данных, которое позволяет выделить более информативные показатели, которые улучшат обучаемость сети. Эти преобразования основываются на опыте и знаниях экспертов в предметной области. Однако в данном случае этот этап был выполнен специалистами компании Microsoft, и научному сообществу были предоставлены уже подготовленные данные.

### 2.1.1. Обнаружение избыточных признаков

Для ИНС избыточность признаков не является проблемой, и не приводит к уменьшению точности предсказаний конечной сети. Однако ИНС очень требовательны к вычислительной мощности и памяти, поэтому устранение лишних атрибутов позволит значительно уменьшить время обучения и размер сети.

Под избыточными признаками понимаются признаки, не обладающие статистической значимостью вследствие:

* критической разрежённости значений признаков,
* значительного перекоса распределения значений признаков,
* высокой степени зависимости от других признаков.

Чтобы удаление каких-то признаков не привело даже к малейшим потерям в эффективности будущей сети, отнесем к избыточным признакам только самые явные, которые имеют более 99%:

* пустых значений,
* одинаковых значений,
* зависимость от другого признака.

В результате применения такого подхода было обнаружено 17 избыточных признаков, которые можно смело не учитывать. Это позволило сократить весь объем исходных данных на 20%. Теперь исходные данные содержат 8 порядковых признаков и 57 номинальных.

### 2.1.2. Подготовка данных

Как уже было отмечено в п. 1.3.5., искусственные нейронные сети накладывают сильные ограничения на входные данные. Так они могут работать только с тензорами (специальными многомерными массивами для числовых данных). Поэтому данные необходимо обработать и поместить в тензоры перед передачей в ИНС.

Нейронные сети не умею обрабатывать пропущенные значения, поэтому необходимо заполнить пропущенные значения признаков. Для этого существует два основных подхода:

* Заполнить недостающие значения нулями, в этом случае они будут игнорироваться сетью;
* Заполнить недостающие значения средним значением – для порядковых признаков, или самой распространенной категорией – для номинальных.

Хорошим решением для ИНС считается заменить недостающие значения нулями, тогда сетью они будут игнорироваться. Главное в этом случае убедиться, что в исходных данных ноль не является осмысленным значением, в этом случае для заполнения пропущенных значений можно взять другое число (например -1).

Для порядковых (числовых) признаков необходимо произвести масштабирование, то есть приведение значений к одному диапазону (обычно это диапазон [0, 1] или [-1, 1]). Это упрощает обучение сети так как из-за непропорционального диапазона значений, разные признаки будут вносить разный вклад в направление обучения сети и будут препятствовать сходимости сети.

Номинальные (категориальные) признаки необходимо векторизовать, для этого используются следующие подходы:

* Кодирование категорий целыми числами в интервале [1, m], где m – число категорий. Применяется как в случае, если категории представлены строками (например ОС компьютера или город), так и в случае числового представления (например номер сборки ОС). Это позволяет равномерно распределить все категории по числовой прямой;
* Прямое кодирование (one-hot encoding). Каждое значение признака кодируется нулевым вектором размера m, с одной единицей на позиции равной номеру соответствующей категории.

На практике можно использовать либо только первый подход, либо оба. В данной работе будут использоваться оба подхода. Использование прямого кодирования часто затруднительно из-за многократного увеличения требуемой памяти, особенно при большом числе категорий. Однако оно дает лучшие результаты благодаря полному отделению категорий друг от друга, так как проецирование номинальных данных на числовую прямую привносит несуществующие связи между категориями.

Чтобы уменьшить размер требуемой памяти при прямом кодировании следует ограничить максимальное число категорий m для всех номинальных признаков. В этом случае нужно оставить только самые популярные категории, а все остальные сделать одной пустой категорией. Само значение m должно быть подобрано экспериментальным путем. Также все данные необходимо разделить на пакеты, размер пакета после кодирования должен соответствовать объему свободной оперативной памяти в процессе обучения.

## 2.2. Типы слоев

Слой – это фундаментальная структура обработки данных в ИНС. Его можно интерпретировать как функцию, которая принимает на вход тензор, и возвращает преобразованный тензор. Преобразование осуществляется, основываясь на значении весовых коэффициентов, типе и размере слоя.

Сначала все слои входящие в ИНС инициализируются небольшими случайными весовыми коэффициентами. На этом этапе сеть не способна выдавать сколько-нибудь значимые предсказания. В процессе обучения весовые коэффициенты оптимизируются, критерием оптимизации является значение функции потерь.

Размер слоя определяется следующими параметрами:

* числом входных нейронов,
* числом внутренних (скрытых) нейронов,
* числом выходных нейронов.

Не существует четких руководств для оптимального выбора этих параметров, поэтому они выбираются экспериментально.

Тип слоя выбирается в зависимости от класса решаемой задачи, он определяет характер внутренних связей нейронов в слое. Часто происходит комбинирование нескольких типов слоев в рамках одной ИНС.

### 2.2.1. Вычислительные слои

Существуют три основных типа вычислительных слоев:

* Полносвязный (dense) слой. Является наиболее распространенным и универсальным. В нем каждый нейрон связан со всеми нейронами предыдущего уровня;
* Сверточный (convolution) слой, в нем каждый нейрон связан только с ограниченным числом нейронов предыдущего уровня, такая связь называется сверткой, благодаря ней ИНС может создавать позиционно независимую иерархию шаблонов. Применяются главным образом для решения задач распознавания образов;
* Рекуррентный (recurrent) слой. Обладает внутренней памятью в отличии от предыдущих слоев. В нем связи между нейронами образуют направленную последовательность, это позволяет обрабатывать данные, в которых присутствует порядковая закономерность. Используются для распознавания речи и текста, машинного перевода и т.д.

Для решаемой задачи применимым является только полносвязный тип слоя, так как он является наиболее универсальным. Тогда как сверточный слой применяется главным образом для работы с изображениями, иногда с музыкой. А рекуррентный применим там, где имеется порядковая зависимость между разными экземплярами данных, например, очередность слов в предложении.

### 2.2.2. Слои преобразования данных

Также существует ряд специальных слоев, использующихся для преобразования данных, которые позволяют решать разного рода задачи, возникающие в процессе обучения ИНС.

Так, для борьбы с переобучением применяется слой прореживания (dropout) [11] который является одним из наиболее распространенных подходов регуляризации ИНС, его можно использовать после каждого вычислительного слоя. Механизм слоя заключается в удалении некоторых, случайно выбираемых атрибутов в процессе обучения сети. Доля удаляемых признаков задается параметром слоя, который называется коэффициент прореживания.

Слой пакетной нормализации (batch normalization) [12] используется для адаптивной нормализации данных между вычислительными слоями. Его принцип работы основан на вычислении экспоненциального скользящего среднего и дисперсии данных, которые используются в процессе обучения сети.

Для уменьшения размерности пространства признаков, то есть для сжатия данных используется слой объединения (pooling). Существует множество вариантов реализации данного слоя, каждый из которых использует свой алгоритм сжатия, например выбор максимального или среднего среди соседних признаков.

Слой активации (activation) преобразует данные в соответствии с функцией активации, которая выбирается в зависимости от требований, которые накладывает решаемая задача. В качестве функции активации может быть, например логистическая (рис. 1.4) или функция ReLU (рис. 2.2). Применение функции активации вносит фактор нелинейности, что расширяет пространство гипотез. Также функцией активации как правило заканчивается ИНС, для того чтобы привести выходные значения к одному диапазону, как правило [0, 1].

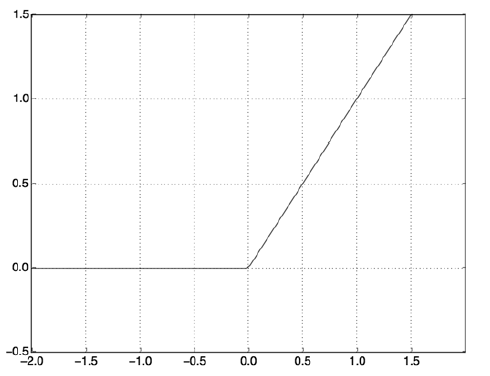


Рисунок 2.2. График функции ReLU [2]

Для выравнивания тензора, то есть преобразования любого тензора к двухмерному с формой (образцы, признаки) применяется слой выпрямления (flatten).

В рамках данной работы планируется использовать все описанные выше слои и исследовать их влияние на итоговую эффективность ИНС.

## 2.3. Функция потерь

Правильный выбор функции потерь играет во многом определяющую роль при проектировании ИНС, так как именно значение этой функции будет минимизировался в процессе обучения. Однако для большинства классических задач существуют конкретные рекомендации по выбору оптимальной функции потерь.

Так для задачи бинарной классификации, в случае, когда результатом работы сети является вероятность, самой подходящей считается функция бинарной перекрестной энтропии (binary cross entropy) [2, с. 97]. Она и будет применятся для реализации ИНС.

## 2.4. Оптимизатор

Функция оптимизации определяет то, как будет изменятся сеть, а именно весовые коэффициенты слоев в процессе обучения. Существуют разные оптимизаторы, каждый из которых реализует конкретный вариант стохастического (случайного) градиентного спуска. Они отличаются различными ухищрениями, которые применяются для борьбы с проблемами, возникающими в процессе обучения, к ним относятся:

* низкая скорость обучения, которая приводит застреванию в локальных минимумах;
* сложная целевая функция, когда сильная нелинейность чередуется с ровными участками;
* неоптимальное обучение на редких, но информативных признаках;
* высокая скорость обучения, которая препятствует сходимости.

Идея градиентного спуска состоит в вычислении градиента в многомерном пространстве признаков, для функции потерь, относительно каждого весового коэффициента. Затем, для уменьшения потерь, требуется незначительно сдвинуть каждый весовой коэффициент в сторону уменьшения градиента.

Наиболее распространенные оптимизаторы это [13]:

* SGD (stochastic gradient descent) – классический градиентный спуск;
* AdaGard (adaptive gradient) – используется идея адаптивного уменьшения скорости обучения для элементов, которые часто обновляются в процессе обучения. То есть чем чаще изменялся некоторый коэффициент, тем больше штраф при последующем изменении. Это позволяет оптимизировать обучение для редко встречающихся, но информативных признаков;
* RMSProp (root mean square propagation) – основан на идее предыдущего алгоритма, но устраняет его главный недостаток, из-за которого, со временем, изменения коэффициентов становятся слишком маленькими, что приводит к параличу обучения. RMSProp может восстанавливать скорость обновления коэффициентов со временем;
* Adam (adaptive moment estimation) – сочетает в себе идею медленного обновления коэффициентов для частых признаков и идею импульса, то есть учитывается не только направление градиента в данной точке, но и недавние изменения данного коэффициента.

В рамках данной работы будет использоваться оптимизатор Adam, так как он совмещает свойства алгоритмов RMSProp и AdaGard, и демонстрирует лучшие результаты.

## 2.5. Топология сети

На первом этапе следует реализовать максимально простую топологию сети, которая будет играть роль базового решения. Эффективность базового решения будет служить точкой отсчета для дальнейших улучшений и усложнений ИНС.

Так как исходные данные содержат:

* Порядковые признаки, которые имеют размерность тензора 2 с формой (образцы, 8);
* Номинальные признаки, с размерностью 3 и формой (образцы, 56, m), где m – число категорий, использующихся при прямом кодировании.

Проектируемая ИНС должна иметь 2 входных слоя для каждого из наборов признаков, которые далее должны объединяться в один полносвязный слой. Однако перед этим их необходимо привести к одной размерности, для этого номинальные признаки необходимо пропустить через flatten слой (п. 2.2.2). А порядковые признаки необходимо нормализовать, для этого они передаются в слой нормализации перед объединением.

Первая версия ИНС будет содержать один полносвязный слой с числом скрытых нейронов равным числу признаков, то есть 65. Эта конфигурация была выбрана интуитивно, далее в главе 4 число слоев и нейронов в них будет экспериментально подобрано для обеспечения наилучшей эффективности.

В конце работы сети необходимо получить вероятность, то есть число в интервале [0, 1], поэтому сеть должна заканчивается слоем активации с функцией sigmoid (рис. 1.4). Так же слои активации могут применяться между полносвязными слоями, при этом можно выбрать из большого числа функции активации, однако именно после последнего слоя функция sigmoid является эталонной. А влияние различных функций активации между полносвязными слоями будет исследовано в главе 4.

Топология описанной сети представлена на рисунке 2.3.

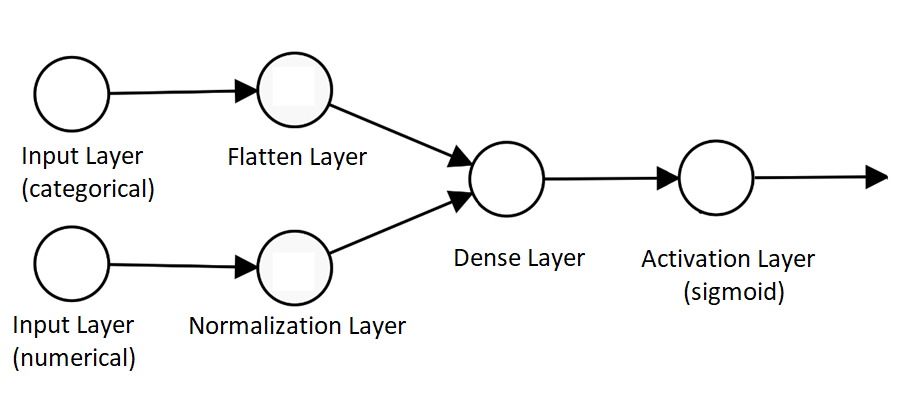


Рисунок 2.3. Схема топологии ИНС

# ГЛАВА 3. ПРОГРАМНАЯ РЕАЛИЗАЦИЯ ИНС

## 3.1. Используемые программные средства

Самым популярным языком программирования, применяемым в сфере машинного обучения, является Python [14], он также очень активно развивается и сейчас находится на первом месте по популярности среди всех языков программирования, основываясь на рейтинге PYPL [15]. Поэтому вокруг языка сформировано огромное сообщество пользователей и написано множество библиотек.

Для разработки искусственных нейронных сетей для языка Python написано достаточно большое число фреймворков, среди которых самые известные это:

* TensorFlow – разрабатывается компанией Google Brain с 2015 года, является самым гибким инструментом позволяющим реализовать любую архитектуру ИНС, требует глубокого понимания математических инструментов, применяемых при обучении ИНС. На данный момент это самый популярный фреймворк для построения ИНС;
* Theano – библиотека реализующая эффективные методы научных вычислений, применяется при решении широкого круга задач машинного обучения (не только ИНС). Разработка ведется с 2007 года группой энтузиастов из Монреальского университета, однако она была приостановлена в 2017 году;
* Caffe – среда глубокого обучения разработанная Яньцинь Цзя в университете Беркли, содержит реализацию множества методов машинного обучения нацеленных главным образом на решение задач классификации и сегментации изображений с упором на ИНС. Используется в множестве больших компаний таких как Yahoo и Facebook, однако на данный момент Caffe считается устаревшим инструментом и активно ведется разработка Caffe2;
* Keras [16] – высокоуровневый интерфейс, представляющий из себя надстройку над фреймворками TensorFlow и Theano. Он позволяет проектировать и реализовывать ИНС достаточно быстро, не вдаваясь глубоко в детали обучения, что идеально подходит для апробации гипотез и создания прототипов. Keras изначально разрабатывался как один из компонентов TensorFlow, однако впоследствии разросся и был выделен в отдельный продукт под руководством инженера Google Франсуа Шолле.

Исходя из вышесказанного, для реализации ИНС был выбран фреймворк Keras, так как он имеет меньший порог вхождения, чем TensorFlow и при этом позволяет строить эффективные решения людям, не являющимся экспертами в области ИНС. Хотя Keras и не позволяет осуществить тонкую настройку ИНС или реализовать неординарные архитектурные подходы, однако этого и не нужно в данной работе.

Помимо фреймворка для построения ИНС Keras, также использовались следующие библиотеки языка Python:

* NumPy [17] – предоставляет вычислительно эффективные алгоритмы обработки данных для языка Python благодаря использованию компилируемых языков;
* pandas [18] и scikit-learn [19] – библиотеки применяющиеся для обработки и анализа данных;
* Mathplotlib [20] – библиотека для визуализации данных (используется для построения графиков).

## 3.1. Предварительная обработка данных

В соответствии с п. 2.1.2. были обработаны исходные данные, код находится в приложении 2. В порядковых признаках пропущенные значения были заполнены нулями и затем все данные были помещены в тензоры, которые могут быть переданы в ИНС.

Номинальные данные были закодированы целыми числами в интервале [1, m], где за максимальное число категорий m было взято число 8 (далее будет подобрано оптимальное число категорий). Затем пропущенные значения признаков также были заполнены нулями, и наконец было применено прямое кодирование (п. 2.1.2).

Замечание: все данные не обрабатываются за одни раз, так как для этого не будет хватать оперативной памяти. Поэтому данные разбиваются на пакеты, каждый из которых обрабатывается непосредственное перед передачей его сети.

## 3.3. Реализация ИНС

На основе описания топологии сети, представленного в п. 2.5. была реализована ИНС на языке Python при помощи фреймворка Keras, код представлен в приложении 3.

В первую очередь был реализован класс Model, позволяющий строить и обучать модель ИНС, при построении модели в нее передаются следующие параметры:

* Число полносвязных слоев и нейронов в каждом из них;
* Наличие слоев активации между полносвязными слоями и имя функции активации;
* Наличие слоев прореживания между полносвязными слоями и коэффициент прореживания;
* Наличие слоев нормализации между полносвязными слоями;
* Нужно ли применять регуляризацию и коэффициент регуляризации;
* Имя используемого оптимизатора.

На данный момент при построении модели ИНС использовался один полносвязный слой без регуляризации, и не использовались слои активации, прореживания и нормализации. В дальнейшем планируется оценить влияние на эффективность каждого из этих параметров и выделить оптимальные, которые обеспечат максимальную эффективность решения.

После построения модели запускается процесс обучения, для этого передаются такие параметры как:

* тренировочные данные;
* данные валидации (они не используются в процессе обучения);
* размер пакета данных, для которого происходит один цикл обучения (256 экземпляров).

Так же был реализован callback класс, который предоставляет возможность вычислять значение площади под ROC-кривой (п. 1.2) в процессе обучения сети (после каждой эпохи). Это необходимо для того, чтобы оценивать динамику обучения, ведь ИНС оптимизирует свои веса для минимизации функции потерь, в нашем случае это функция binary cross entropy. Однако это не значит, что значение целевой функции (площади под ROC-кривой) будет увеличиваться, поэтому ее значение необходимо контролировать в процессе обучения. Заметим, что функцию площади под ROC-кривой нельзя использовать в качестве функции потерь, так как она вычисляется для всего набора данных, тогда как функция потерь вычисляется для пакета данных, который использовался в данном цикле обучения.

На рисунке 3.1. представлена динамика обучения реализованной ИНС на протяжении 26 эпох. Каждая эпоха включает в себя обучение сети на 65536-ти новых экземпляров компьютеров. По вертикали отложено значение площади под ROC-кривой, а по горизонтали – эпохи. Точками отмечены результаты для тренировочных данных, а сплошной линией – для данных, которые не участвовали в процессе обучения. Как можно заметить сеть не переобучилась и при этом достаточно быстро достигла предельной точности предсказаний. Это объясняется маленьким размером сети, т.к. она содержит всего один полносвязный слой с 457 параметрами (весовыми коэффициентами).

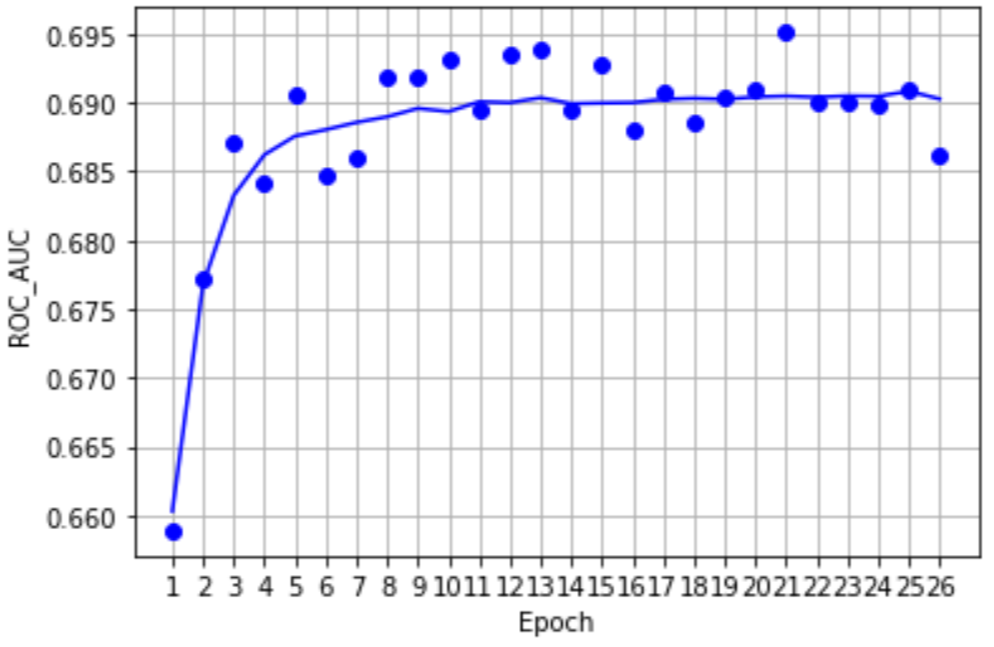


Рисунок 3.1. Динамика обучения ИНС

Для более точной и объективной оценки эффективности полученного решения необходимо сгенерировать предсказания реализованной ИНС для тестовой выборки. Далее полученные предсказания отправляются на сервер соревнования [3], где имеются правильные ответы, на их основе и вычисляется, а затем отображается значение площади под ROC-кривой для тестовой выборки. Эффективность ИНС теперь можно сравнить с решениями других исследователей.

Итак, реализованная ИНС получила оценку качества классификации равную 0,573, тогда как пять лучших решений на данный момент показывают оценки в пределах 0,664-0,676. Полученный результат демонстрирует применимость использования ИНС для решения поставленной задачи, так как даже простое архитектурное решение обладает достаточно неплохой эффективность классификации по сравнению с лучшими решениями. Следующим шагом планируется усложнить архитектуру ИНС для повышения качества классификации.

# ГЛАВА 4. ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ ИНС

В данной главе планируется усовершенствовать архитектуру ИНС для того, чтобы улучшить качество классификации и приблизить ее эффективность к лучшим существующим решениям. Для этого планируется подобрать следующие параметры сети:

* Количество полносвязных слоев и нейронов в каждом из них;
* Наличие слоев активации между полносвязными слоями и имя функции активации;
* Наличие слоев прореживания между полносвязными слоями и коэффициент прореживания;
* Наличие слоев нормализации между полносвязными слоями;
* Нужно ли применять регуляризацию и коэффициент регуляризации.

На первом этапе необходимо исследовать слои активации и нормализации, так как от них во многом будут зависеть результаты дальнейших экспериментов. На следующем шаге будет подобрано число полносвязных слоев и их размер, для обеспечения максимальной точности ИНС. И уже в конце будет применена регуляризация и прореживание, для борьбы с переобучением сети.

## 4.1. Применение слоев активации и нормализации

Для оценки влияния слоя активации на эффективность решения в исходную ИНС (рис.2.3) перед единственным полносвязным слоем был добавлен еще один с 64-мя скрытыми нейронами. При этом это практически не отразилось на динамике обучения по сравнению с исходной сетью (рис.4.1).

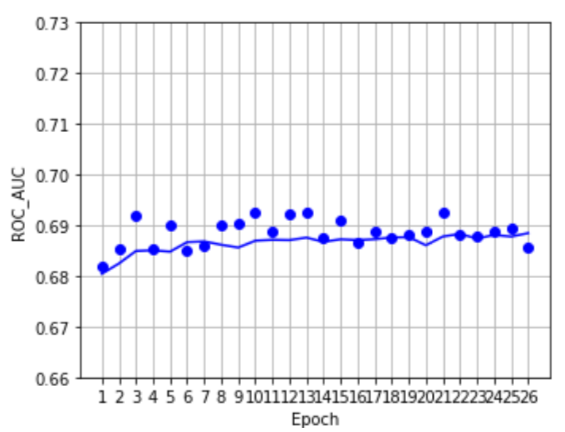
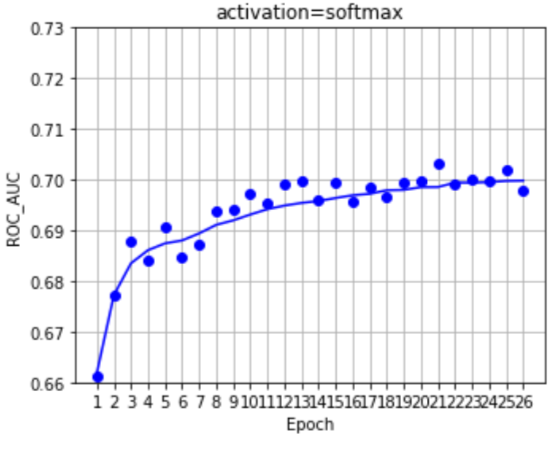
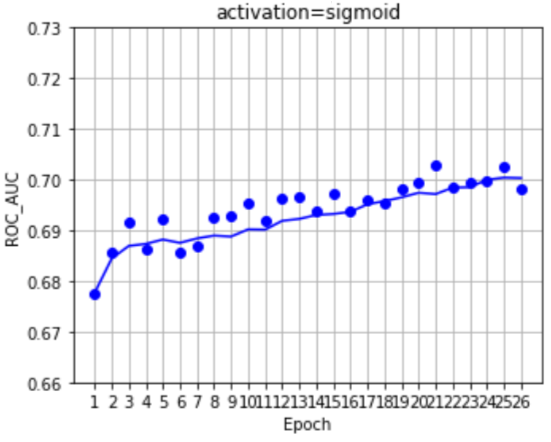


Рисунок 4.1. Динамика обучения ИНС с двумя полносвязными слоями

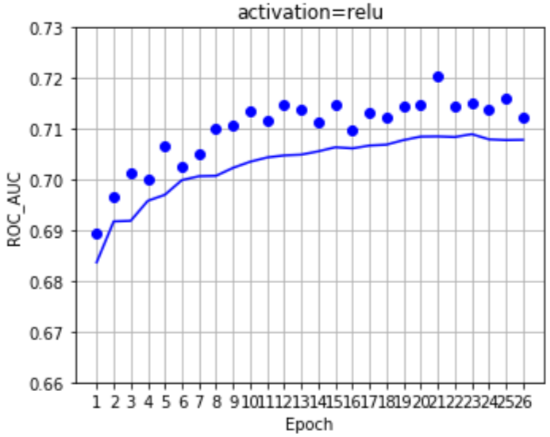
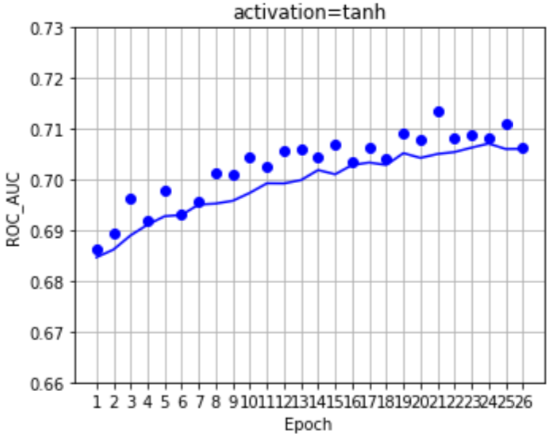
Далее был добавлен слой активации между полносвязными слоями и затем построено и обучено 4 сети, в каждой из которых применялась своя функция активации (рис.4.2), а именно: softmax, sigmoid, relu, tanh.

b)

b)

a)

d)

c)

Рисунок 4.2. Динамика обучения ИНС с разными функциями активации: *а* – softmax; *b* – sigmoid; *c* – relu; *d* – tanh

Можно заметить, что применение любой функции активации значительно повышает эффективность обучения сети. Однако самый значительный прирост продемонстрировала функция relu, поэтому в дальнейших реализациях ИНС, после каждого полносвязного слоя, будет применятся слой активации с этой функцией.

Применение же слоя нормализации не продемонстрировало каких-либо улучшений при обучении ИНС (рис.4.3), даже наоборот, качество классификации незначительно снизилось (по сравнению с рис.4.2с). Это можно объяснить тем, что большая часть признаков в обучающих данных является категориальными, которые не нуждаются в нормализации, а порядковые признаки уже были нормализованы ранее.

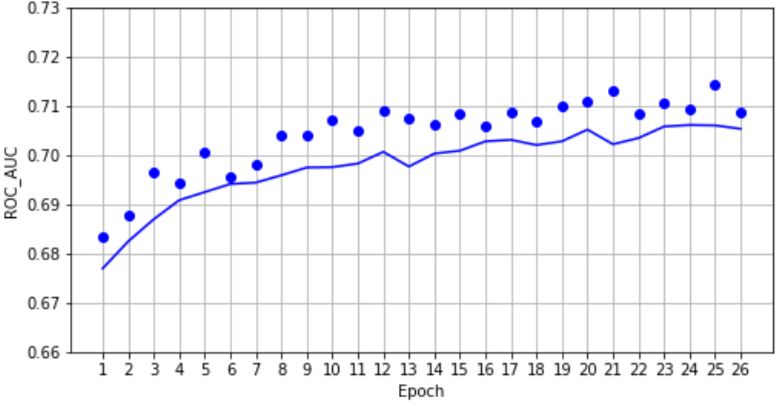


Рисунок 4.4. Динамика обучения ИНС с применением слоя нормализации

В итоге ИНС, с применением функции активации relu и без нормализации, на тестовой выборке продемонстрировала значительные улучшения (по сравнению с предыдущим 0,573), показав результат 0,602.

## 4.2. Подбор размера полносвязных слоев и глубины ИНС

На следующем этапе необходимо подобрать оптимальное число полносвязных слоев и их размер. Для этого было выбрано два параметра: максимальное число слоев результирующей сети, равное трем; и восемь вариантов различных размеров полносвязного слоя (0, 16, 32, 64, 128, 256, 512, 1024), где 0 означает отсутствие слоя. Под размером слоя понимается число нейронов в слое. Методика выбора лучшей конфигурации осуществилась простым перебором, это довольно примитивный метод, зато он позволяет наверняка найти лучшее решение. То есть было обучено 24 ИНС, где для каждого из трех слоев было перебрано 8 различных вариантов размера слоя.

Динамика обучения для различных вариантов размера первого слоя представлена на рис.4.5. Можно отметить, что значительно отличаются только сети без нового слоя (neurons\_number=0), и сеть с добавлением минимального слоя (neurons\_number=16). Далее, при увеличении числа нейронов в слое, сеть незначительно повышает качество предсказаний. Однако так же возрастает разрыв между точностью предсказаний на проверочных данных (сплошная линия) и на обучающих (пунктирная линия). Это свидетельствует стремлении сети к переобучению, при увеличении размера слоя.

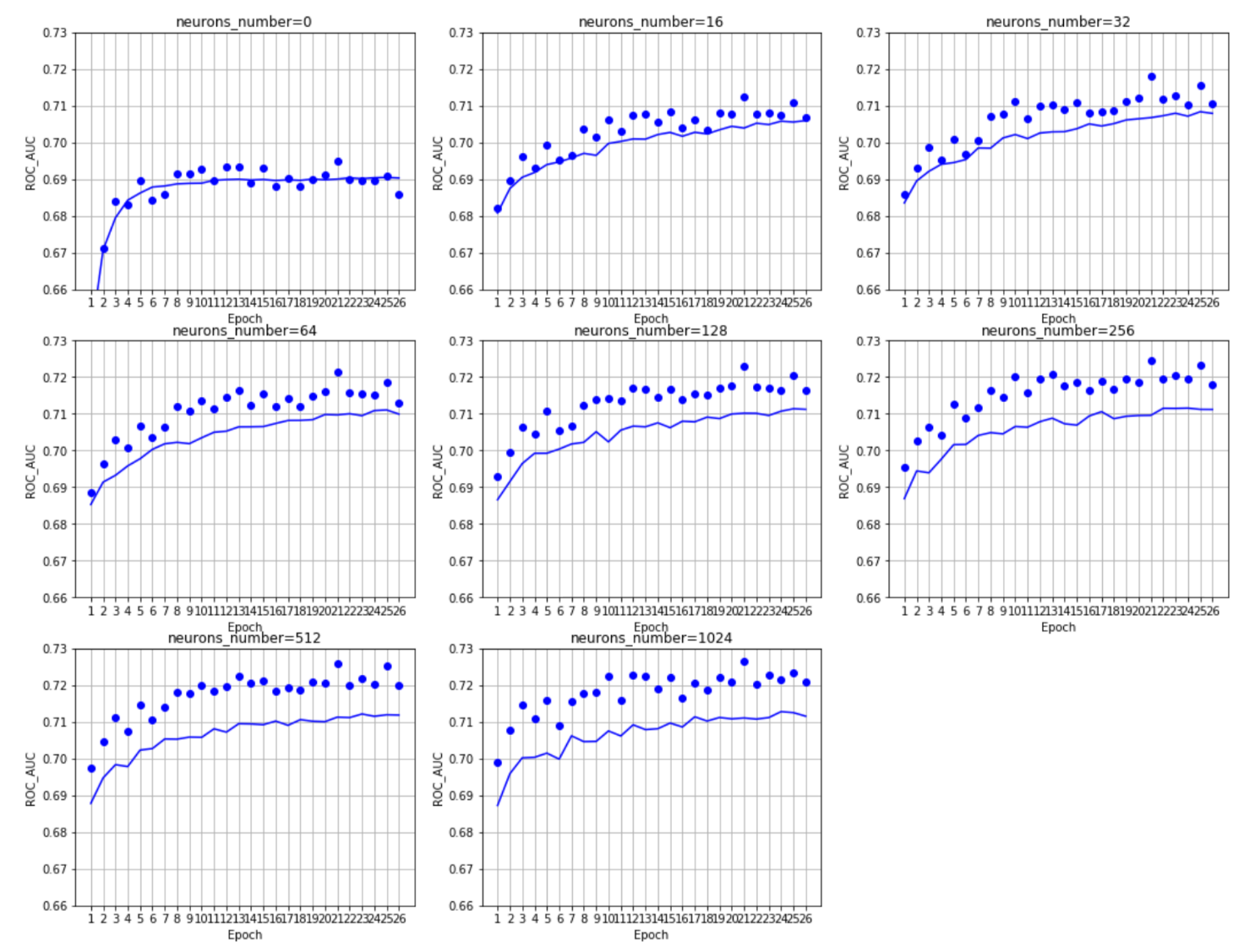


Рисунок 4.4. Динамика обучения ИНС для различных размеров первого слоя

Добавление второго слоя практчески не сказалось на обучаемости сети (рис.4.5). Совершенно аналогичным образов выглядят графики для третьего слоя, поэтому они небыли включены. Исходя из этого можно сделать вывод, что для достижения максимальной эффективности сети достаточно добавить один полносвязный слой с 64-мя скрытыми нейронами. Такой слой обеспечивает оптимальное соотношение между качеством предсказаний и переобучением сети.

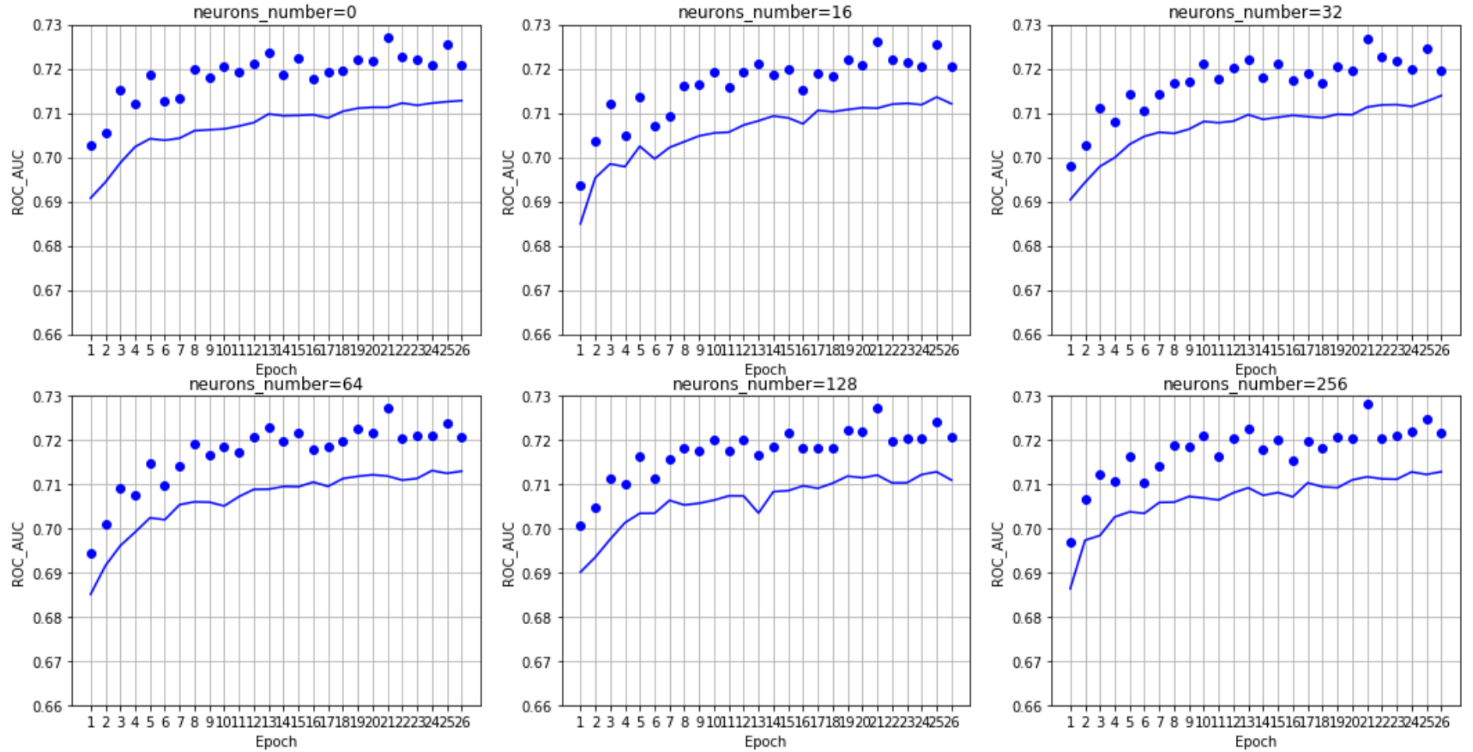


Рисунок 4.5. Динамика обучения ИНС для различных размеров второго слоя

На тестовых данных, лучшая полученная в рамках данного параграфа сеть незначительно улучшила предыдущий результат, продемонстрировав значение 0,607.

## 4.3. Применение регуляризации и прореживания

Методы прореживания и регуляризации позволяют бороться с переобучением ИНС. Однако они ухудшают результаты на обучающих данных, поэтому их влияние на итоговый результат достаточно сложно оценить, ведь для каждой конфигурации сети необходимо будет вычислять значения ROC\_AUC метрики для всего тестового набора данных.

## 4.4. Итоговая конфигурация ИНС

В результате проведенных экспериментов была подобрана такая комбинация гиперпараметров ИНС, которая позволила значительно улучшить эффективность сети, по сравнению с базовым решением, представленным в п.3.3. Схема топологии полученной ИНС представлена на рисунке 4.6. На тестовых данных она демонстрирует результат ROC\_AUC метрики, равный 0,652. Это значение входит в топ 30 лучших решений в рамках соревнования от Microsoft. [3]

# ЗАКЛЮЧЕНИЕ

В настоящей работе была спроектирована, реализована и обучена ИНС, способная решать задачу оценки рисков заражения компьютера вредоносным ПО. Для улучшения качества предсказаний сети, путем эксперимента, была подобрана оптимальная комбинация гиперпараметров модели ИНС. Это позволило получить решение, которое эффективно справляется с поставленной задачей, на ряду лучшими решениями, существующими на данный момент.

Для дальнейшего повышения качества предсказаний можно выделить следующие направления работы:

* Использовать более продвинутые фреймворки для конструирования ИНС, такие как TensorFlow, которые позволяют осуществлять более тонкую настройку сети и реализовывать нестандартные архитектурные решения;
* Применять ансамбли решений, полученных с помощью различных методов машинного обучения. Большая часть самых эффективных реализаций получены именно благодаря применению такого подхода. Потому что все методы имеют свои слабые и сильные стороны, которые при объединении взаимодополняют друг друга.

Однако это все равно не позволит добиться значительного улучшения полученного результата, так как

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Годфеллоу Я. Глубокое обучение / И. [Бенджио](https://www.labirint.ru/authors/190834/), Я. [Гудфеллоу](https://www.labirint.ru/authors/190835/), А. [Курвилль](https://www.labirint.ru/authors/190836/); пер. с англ. А.А. Слинкин, ред. Д.А. Мовчан. 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.
2. Шолле Ф. Глубокое обучение на Python – СПб.: Питер, 2018 – 400 с.
3. Microsoft Malware Prediction Competition / [Электронный документ] (https://www.kaggle.com/c/microsoft-malware-prediction). Проверено 20.03.2019.
4. AUC ROC (площадь под кривой ошибок) / [Электронный документ] (https://dyakonov.org/2017/07/28/auc-roc-площадь-под-кривой-ошибок/). Проверено 03.03.2019.
5. Деревья решений – общие принципы работы / [Электронный документ] (https://basegroup.ru/community/articles/description). Проверено 16.04.2019.
6. Метод деревьев решений для задачи классификации / [Электронный документ] (https://edu.kpfu.ru/pluginfile.php/91556/mod\_resource/content /2/Decision%20trees\_1.pdf). Проверено 16.04.2019.
7. Чистяков С.П. Случайные леса: обзор / C.П. Чистяков – Труды Карельского научного центра РАН – № 1. – 2013. – С. 117-136; [Электронный документ] (http://resources.krc.karelia.ru/transactions/doc/trudy2013/trudy\_2013\_1\_ 117-136.pdf). Проверено 20.05.2019.
8. Цыплаков А.А. Некоторые эконометрические методы. Метод максимального правдоподобия в эконометрии / Методическое пособие. – Новосибирск: НГУ, 1997. – 127 С.
9. Воронцов К.В. Лекции по линейным алгоритмам классификации / [Электронный документ] (http://www.machinelearning.ru/wiki/images/6 /68/voron-ML-Lin.pdf). Проверено 18.04.2019.
10. Хайкин С. Нейронные сети: полный курс, 2-е изд.: Пер. с англ. – М.: Вильямс, 2006 – 1104 с.
11. Dropout: A Simple Way to Prevent Neural Networks from Overfitting / G.E. Hinton [и др.] // Journal of Machine Learning Research – 2014. – № 15 – C. 1929-1958; [Электронный документ] (http://jmlr.org/papers/volume15/srivastava14a. old/srivastava14a.pdf). Проверено 20.05.2019.
12. Ioffe S. // Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift / S. Ioffe, C. Szegedy // Proceedings of the 32nd International Conference on Machine Learning – Lille, 2015. – C. 448-456; [Электронный документ] (https://arxiv.org/pdf/1502.03167.pdf). Проверено 20.05.2019.
13. Садовников П. Методы оптимизации нейронных сетей / [Электронных документ] (https://habr.com/ru/post/318970/). Проверено 09.05.2019.
14. Python / [Электронный документ] (https://www.python.org/). Проверено. 13.05.2019.
15. PopularitY of Programming Language / [Электронный документ] (http://pypl.github.io/PYPL.html). Проверено. 13.05.2019.
16. Keras Documentation. / [Электронный документ] (https://keras.io/). Проверено 13.03.2019.
17. NumPy / [Электронный документ] (http://www.numpy.org/). Проверено. 13.05.2019.
18. Python Data Analysis Library / [Электронный документ] (https://pandas.pydata.org/). Проверено. 13.05.2019.
19. scikit-learn: machine learning in Python / [Электронный документ] (https://scikit-learn.org/stable/documentation.html). Проверено 03.03.2019.
20. Matplotlib: Python plotting / [Электронный документ] (https://matplotlib.org/). Проверено. 13.05.2019.

Приложение 1

**Набор признаков в исходных данных**

Unavailable or self-documenting column names are marked with an "NA".

* MachineIdentifier - Individual machine ID
* ProductName - Defender state information e.g. win8defender
* EngineVersion - Defender state information e.g. 1.1.12603.0
* AppVersion - Defender state information e.g. 4.9.10586.0
* AvSigVersion - Defender state information e.g. 1.217.1014.0
* IsBeta - Defender state information e.g. false
* RtpStateBitfield - NA
* IsSxsPassiveMode - NA
* DefaultBrowsersIdentifier - ID for the machine's default browser
* AVProductStatesIdentifier - ID for the specific configuration of a user's antivirus software
* AVProductsInstalled - NA
* AVProductsEnabled - NA
* HasTpm - True if machine has tpm
* CountryIdentifier - ID for the country the machine is located in
* CityIdentifier - ID for the city the machine is located in
* OrganizationIdentifier - ID for the organization the machine belongs in, organization ID is mapped to both specific companies and broad industries
* GeoNameIdentifier - ID for the geographic region a machine is located in
* LocaleEnglishNameIdentifier - English name of Locale ID of the current user
* Platform - Calculates platform name (of OS related properties and processor property)
* Processor - This is the process architecture of the installed operating system
* OsVer - Version of the current operating system
* OsBuild - Build of the current operating system
* OsSuite - Product suite mask for the current operating system.
* OsPlatformSubRelease - Returns the OS Platform sub-release (Windows Vista, Windows 7, Windows 8, TH1, TH2)
* OsBuildLab - Build lab that generated the current OS. Example: 9600.17630.amd64fre.winblue\_r7.150109-2022
* SkuEdition - The goal of this feature is to use the Product Type defined in the MSDN to map to a 'SKU-Edition' name that is useful in population reporting. The valid Product Type are defined in %sdxroot%\data\windowseditions.xml. This API has been used since Vista and Server 2008, so there are many Product Types that do not apply to Windows 10. The 'SKU-Edition' is a string value that is in one of three classes of results. The design must hand each class.
* IsProtected - This is a calculated field derived from the Spynet Report's AV Products field. Returns: a. TRUE if there is at least one active and up-to-date antivirus product running on this machine. b. FALSE if there is no active AV product on this machine, or if the AV is active, but is not receiving the latest updates. c. null if there are no Anti Virus Products in the report. Returns: Whether a machine is protected.
* AutoSampleOptIn - This is the SubmitSamplesConsent value passed in from the service, available on CAMP 9+
* PuaMode - Pua Enabled mode from the service
* SMode - This field is set to true when the device is known to be in 'S Mode', as in, Windows 10 S mode, where only Microsoft Store apps can be installed
* IeVerIdentifier - NA
* SmartScreen - This is the SmartScreen enabled string value from registry. This is obtained by checking in order, HKLM\SOFTWARE\Policies\Microsoft\Windows\System\SmartScreenEnabled and HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\SmartScreenEnabled. If the value exists but is blank, the value "ExistsNotSet" is sent in telemetry.
* Firewall - This attribute is true (1) for Windows 8.1 and above if windows firewall is enabled, as reported by the service.
* UacLuaenable - This attribute reports whether or not the "administrator in Admin Approval Mode" user type is disabled or enabled in UAC. The value reported is obtained by reading the regkey HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\EnableLUA.
* Census\_MDC2FormFactor - A grouping based on a combination of Device Census level hardware characteristics. The logic used to define Form Factor is rooted in business and industry standards and aligns with how people think about their device. (Examples: Smartphone, Small Tablet, All in One, Convertible...)
* Census\_DeviceFamily - AKA DeviceClass. Indicates the type of device that an edition of the OS is intended for. Example values: Windows.Desktop, Windows.Mobile, and iOS.Phone
* Census\_OEMNameIdentifier - NA
* Census\_OEMModelIdentifier - NA
* Census\_ProcessorCoreCount - Number of logical cores in the processor
* Census\_ProcessorManufacturerIdentifier - NA
* Census\_ProcessorModelIdentifier - NA
* Census\_ProcessorClass - A classification of processors into high/medium/low. Initially used for Pricing Level SKU. No longer maintained and updated
* Census\_PrimaryDiskTotalCapacity - Amount of disk space on primary disk of the machine in MB
* Census\_PrimaryDiskTypeName - Friendly name of Primary Disk Type - HDD or SSD
* Census\_SystemVolumeTotalCapacity - The size of the partition that the System volume is installed on in MB
* Census\_HasOpticalDiskDrive - True indicates that the machine has an optical disk drive (CD/DVD)
* Census\_TotalPhysicalRAM - Retrieves the physical RAM in MB
* Census\_ChassisTypeName - Retrieves a numeric representation of what type of chassis the machine has. A value of 0 means xx
* Census\_InternalPrimaryDiagonalDisplaySizeInInches - Retrieves the physical diagonal length in inches of the primary display
* Census\_InternalPrimaryDisplayResolutionHorizontal - Retrieves the number of pixels in the horizontal direction of the internal display.
* Census\_InternalPrimaryDisplayResolutionVertical - Retrieves the number of pixels in the vertical direction of the internal display
* Census\_PowerPlatformRoleName - Indicates the OEM preferred power management profile. This value helps identify the basic form factor of the device
* Census\_InternalBatteryType - NA
* Census\_InternalBatteryNumberOfCharges - NA
* Census\_OSVersion - Numeric OS version Example - 10.0.10130.0
* Census\_OSArchitecture - Architecture on which the OS is based. Derived from OSVersionFull. Example - amd64
* Census\_OSBranch - Branch of the OS extracted from the OsVersionFull. Example - OsBranch = fbl\_partner\_eeap where OsVersion = 6.4.9813.0.amd64fre.fbl\_partner\_eeap.140810-0005
* Census\_OSBuildNumber - OS Build number extracted from the OsVersionFull. Example - OsBuildNumber = 10512 or 10240
* Census\_OSBuildRevision - OS Build revision extracted from the OsVersionFull. Example - OsBuildRevision = 1000 or 16458
* Census\_OSEdition - Edition of the current OS. Sourced from HKLM\Software\Microsoft\Windows NT\CurrentVersion@EditionID in registry. Example: Enterprise
* Census\_OSSkuName - OS edition friendly name (currently Windows only)
* Census\_OSInstallTypeName - Friendly description of what install was used on the machine i.e. clean
* Census\_OSInstallLanguageIdentifier - NA
* Census\_OSUILocaleIdentifier - NA
* Census\_OSWUAutoUpdateOptionsName - Friendly name of the WindowsUpdate auto-update settings on the machine.
* Census\_IsPortableOperatingSystem - Indicates whether OS is booted up and running via Windows-To-Go on a USB stick.
* Census\_GenuineStateName - Friendly name of OSGenuineStateID. 0 = Genuine
* Census\_ActivationChannel - Retail license key or Volume license key for a machine.
* Census\_IsFlightingInternal - NA
* Census\_IsFlightsDisabled - Indicates if the machine is participating in flighting.
* Census\_FlightRing - The ring that the device user would like to receive flights for. This might be different from the ring of the OS which is currently installed if the user changes the ring after getting a flight from a different ring.
* Census\_ThresholdOptIn - NA
* Census\_FirmwareManufacturerIdentifier - NA
* Census\_FirmwareVersionIdentifier - NA
* Census\_IsSecureBootEnabled - Indicates if Secure Boot mode is enabled.
* Census\_IsWIMBootEnabled - NA
* Census\_IsVirtualDevice - Identifies a Virtual Machine (machine learning model)
* Census\_IsTouchEnabled - Is this a touch device ?
* Census\_IsPenCapable - Is the device capable of pen input
* Census\_IsAlwaysOnAlwaysConnectedCapable - Retreives information about whether the battery enables the device to be AlwaysOnAlwaysConnected .
* Wdft\_IsGamer - Indicates whether the device is a gamer device or not based on its hardware combination.
* Wdft\_RegionIdentifier - NA

Приложение 2

**Код предварительной обработки исходных данных**

import gc

import numpy as np

import pandas as pd

from tqdm import tqdm

from datetime import datetime

from sklearn.preprocessing import LabelEncoder

*# Global params*

train\_size = 4194304

test\_size = None

valid\_rate = 0.2

*# 1. Data types*

dtypes = {

'MachineIdentifier': 'category',

'ProductName': 'category',

'EngineVersion': 'category',

'AppVersion': 'category',

'AvSigVersion': 'category',

'RtpStateBitfield': 'float16',

'IsSxsPassiveMode': 'int8',

'DefaultBrowsersIdentifier': 'float32',

'AVProductStatesIdentifier': 'float32',

'AVProductsInstalled': 'float16',

'AVProductsEnabled': 'float16',

'HasTpm': 'int8',

'CountryIdentifier': 'int16',

'CityIdentifier': 'float32',

'OrganizationIdentifier': 'float16',

'GeoNameIdentifier': 'float16',

'LocaleEnglishNameIdentifier': 'int16',

'OsVer': 'category',

'OsBuild': 'int16',

'OsSuite': 'int16',

'OsPlatformSubRelease': 'category',

'OsBuildLab': 'category',

'SkuEdition': 'category',

'IsProtected': 'float16',

'IeVerIdentifier': 'float16',

'SmartScreen': 'category',

'Firewall': 'float16',

'Census\_MDC2FormFactor': 'category',

'Census\_OEMNameIdentifier': 'float32',

'Census\_OEMModelIdentifier': 'float32',

'Census\_ProcessorCoreCount': 'float16',

'Census\_ProcessorManufacturerIdentifier': 'float16',

'Census\_ProcessorModelIdentifier': 'float32',

'Census\_PrimaryDiskTotalCapacity': 'float64',

'Census\_PrimaryDiskTypeName': 'category',

'Census\_SystemVolumeTotalCapacity': 'float64',

'Census\_HasOpticalDiskDrive': 'int8',

'Census\_TotalPhysicalRAM': 'float32',

'Census\_ChassisTypeName': 'category',

'Census\_InternalPrimaryDiagonalDisplaySizeInInches':'float32',

'Census\_InternalPrimaryDisplayResolutionHorizontal':'float32',

'Census\_InternalPrimaryDisplayResolutionVertical': 'float32',

'Census\_PowerPlatformRoleName': 'category',

'Census\_InternalBatteryType': 'category',

'Census\_InternalBatteryNumberOfCharges': 'float64',

'Census\_OSVersion': 'category',

'Census\_OSArchitecture': 'category',

'Census\_OSBranch': 'category',

'Census\_OSBuildNumber': 'int16',

'Census\_OSBuildRevision': 'int32',

'Census\_OSEdition': 'category',

'Census\_OSInstallTypeName': 'category',

'Census\_OSUILocaleIdentifier': 'int16',

'Census\_OSWUAutoUpdateOptionsName': 'category',

'Census\_GenuineStateName': 'category',

'Census\_ActivationChannel': 'category',

'Census\_FlightRing': 'category',

'Census\_FirmwareManufacturerIdentifier': 'float16',

'Census\_FirmwareVersionIdentifier': 'float32',

'Census\_IsSecureBootEnabled': 'int8',

'Census\_IsTouchEnabled': 'int8',

'Census\_IsPenCapable': 'int8',

'Census\_IsAlwaysOnAlwaysConnectedCapable': 'float16',

'Wdft\_IsGamer': 'float16',

'Wdft\_RegionIdentifier': 'float16',

'HasDetections': 'int8'

}

true\_numerical\_columns = [

'Census\_ProcessorCoreCount',

'Census\_PrimaryDiskTotalCapacity',

'Census\_SystemVolumeTotalCapacity',

'Census\_TotalPhysicalRAM',

'Census\_InternalPrimaryDiagonalDisplaySizeInInches',

'Census\_InternalPrimaryDisplayResolutionHorizontal',

'Census\_InternalPrimaryDisplayResolutionVertical',

'Census\_InternalBatteryNumberOfCharges'

]

*# 2. Loading data*

retained\_columns = list(dtypes.keys())

train = pd.read\_csv('../input/microsoft-malware-prediction/train.csv',

nrows = train\_size,

usecols = retained\_columns,

dtype = dtypes)

retained\_columns.remove('HasDetections')

test = pd.read\_csv('../input/microsoft-malware-prediction/test.csv',

nrows = test\_size,

usecols = retained\_columns,

dtype = dtypes)

retained\_columns.remove('MachineIdentifier')

target = train['HasDetections'].values

del train['HasDetections'], train['MachineIdentifier'], test['MachineIdentifier']

gc.collect()

*# 3. Сreating a dictionary of categorical data*

cat\_dict = {}

for i **in** tqdm(retained\_columns):

comb = pd.concat((train[i], test[i]), axis=0)

cat\_dict[i] = comb.value\_counts().keys().tolist()

if str(train[i].dtype) != 'category':

cat\_dict[i] = [round(x, 1) for x **in** cat\_dict[i]]

cat\_dict[i] = np.array(cat\_dict[i]).astype(str)

*# 3.1. One-hot encoding categorical data*

def categorical(data, n):

result = np.zeros((len(data), 1, n))

for i **in** data.keys():

if i **not** **in** true\_numerical\_columns:

dictionary = cat\_dict[i]

tmp = data[i].values.astype(str)

if data[i].isnull().values.any() **or** n <= len(dictionary):

if n <= len(dictionary):

enc\_dict = {c: 'nan' for c **in** dictionary[n-1:]}

dictionary = dictionary[:n-1]

vfunc = np.vectorize(lambda x: enc\_dict.get(x, x))

tmp = vfunc(tmp)

dictionary = np.append(dictionary, 'nan')

le = LabelEncoder().fit(dictionary)

tmp = le.transform(tmp)

to\_categ = utils.to\_categorical(tmp, num\_classes=n).reshape(len(tmp), 1, n)

result = np.concatenate((result, to\_categ), axis=1)

gc.collect()

return result[:, 1:]

*# 3.2. Processing numerical data*

def numerical(data):

result = np.zeros((len(data), 1))

for i **in** true\_numerical\_columns:

data[i] = data[i].fillna(0)

result = np.concatenate((result, np.array(data[i]).reshape(len(data), 1)), axis=1)

return result[:, 1:]

*# 4. Selecting validation data*

split = round(len(train) \* (1 - valid\_rate))

valid = [numerical(train[split:]), categorical(train[split:], categories)]

train = train[:split]

target\_train = target[:split]

target\_valid = target[split:]

Приложение 3

**Код реализованной модели ИНС на языке Python**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from keras import models, regularizers, utils

from keras.callbacks import Callback

from keras.layers import Input, Dense, concatenate, Activation, BatchNormalization, Flatten, Dropout, Embedding

*# Global params*

batch\_size = 256

categories = 8

step = 65536

params = []

activation = False

normalization=False

dropout=False

regularization=False

optimizer='Adam'

*# ROC AUC Callback class*

class **roc\_auc\_callback**(Callback):

def \_\_init\_\_(self, train, valid):

self.x\_train = train[0]

self.y\_train = train[1]

self.x\_valid = valid[0]

self.y\_valid = valid[1]

self.roc\_max = 0

def on\_epoch\_end(self, epoch, logs={}):

y\_pred = self.model.predict(self.x\_train)

roc\_train = roc\_auc\_score(self.y\_train, y\_pred)

y\_pred\_valid = self.model.predict(self.x\_valid)

roc\_valid = roc\_auc\_score(self.y\_valid, y\_pred\_valid)

print('roc: **%s** - roc-val: **%s**' % (str(round(roc\_train, 4)),str(round(roc\_valid, 4))), end=100\*' '+'**\n**')

if self.roc\_max < roc\_valid:

self.roc\_max = roc\_valid

self.model.save('model.h5')

logs['roc\_val'] = roc\_valid

logs['roc'] = roc\_train

return

*# Neural Network model class*

class **Model**:

def \_\_init\_\_(self, params, categories=8, activation=False, dropout=False, normalization=False, regularization=False, optimizer='Adam'):

input\_numerical = Input(shape=(8,))

normalization\_numerical = BatchNormalization()(input\_numerical)

input\_categorical = Input(shape=(56, categories,))

flatten\_categorical = Flatten()(input\_categorical)

x = concatenate([normalization\_numerical, flatten\_categorical])

for layer in params:

if regularization:

x = Dense(layer, kernel\_regularizer=regularizers.l1(regularization))(x)

else:

x = Dense(layer)(x)

if activation:

x = Activation(activation)(x)

if dropout:

x = Dropout(dropout)(x)

if normalization:

x = BatchNormalization()(x)

output\_layer = Dense(1, activation='sigmoid')(x)

self.model = models.Model(inputs=(input\_numerical, input\_categorical), outputs=output\_layer)

self.model.compile(optimizer=optimizer, loss='binary\_crossentropy')

def fit(self, train, valid, epochs=1, batch\_size=256):

return self.model.fit(train[0], train[1],

validation\_data=valid,

epochs=epochs,

shuffle=True,

verbose=1,

callbacks=[roc\_auc\_callback(train, valid)],

batch\_size=batch\_size)

def summary(self):

return self.model.summary()

*# Build model*

model = Model(categories, activation=activation, normalization=normalization, dropout=dropout, regularization=regularization, optimizer=optimizer)

model.summary()

*# Train model*

plot = {'roc': [], 'roc\_val': []}

for i **in** range(0, len(train), step):

tmp\_c = categorical(train[i:i + step], categories)

tmp\_n = numerical(train[i:i + step])

history = model.fit(([tmp\_n, tmp\_c], target\_train[i:i + step]),

(valid, target\_valid),

batch\_size=batch\_size)

plot['roc'].append(history.history['roc'][0])

plot['roc\_val'].append(history.history['roc\_val'][0])

*# Plot training results*

plt.plot(range(1, len(plot['roc'])+1), plot['roc'], 'bo')

plt.plot(range(1, len(plot['roc'])+1), plot['roc\_val'], 'b')

plt.title('Traning progress')

plt.xlabel('Epoch')

plt.ylabel('ROC')

plt.xticks(range(1, len(plot['roc'])+1))

plt.legend()

plt.grid(True)

plt.show()

*# Predict test results*

pred = []

model = models.load\_model('model.h5')

for i **in** range(0, len(test), step):

tmp\_c = categorical(test[i:i + step], categories)

tmp\_n = numerical(test[i:i + step])

pred[i:i+step] = model.predict([tmp\_n, tmp\_c])

pred = [pred[i][0] for i **in** range(len(pred))]